PART B – DESIGN DOCUMENT + PROJECT EXPLANATION

1. Project Overview

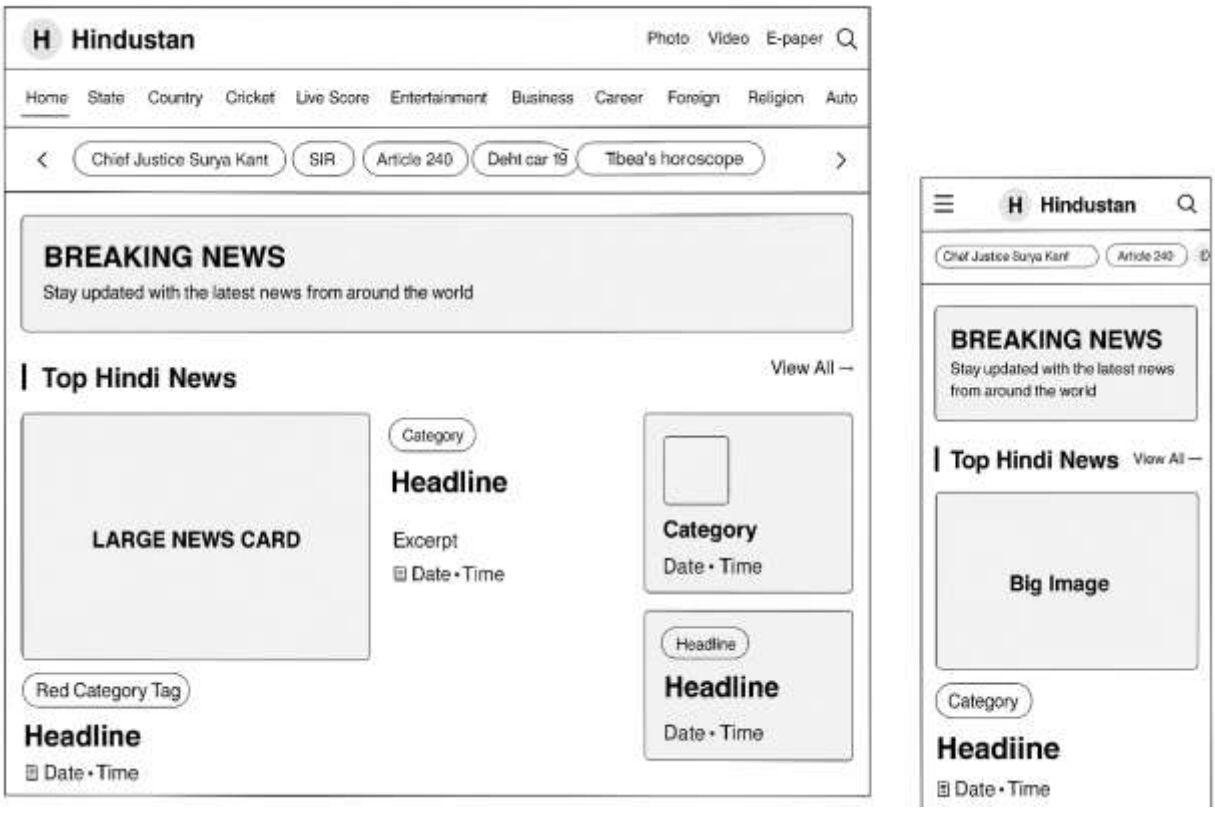
This project is a **Front-Page Clone of LiveHindustan.com**, built using **Next.js**, **TypeScript**, and **TailwindCSS**.
It includes:

- Homepage with hero news, breaking news, trending topics
- Category news pages (`/[category]`)
- Dynamic article pages (`/news/[id]`)
- Multiple reusable UI components
- Fully responsive mobile and desktop layout
- Data fetched from mock JSON or real news API
- Optimized images using Next.js `<Image>`

This clone replicates the structure, layout style, and UX patterns of modern Hindi news portals.

2. Wireframes

Wireframe



These wireframes match your UI precisely — with large hero card, trending pills, section headers, and responsive layouts.

---

3. Layout Decisions (Why This Design?)

✓ A. Top Navigation + Categories

- Placed horizontally for quick access to the main sections.
- Matches real newspapers like LiveHindustan, NDTV, AajTak.
- On mobile, collapses into hamburger menu to save space.

✓ B. Trending Topics (Chips)

- Placed below navbar, scrollable horizontally.
- Helps highlight trending searches.
- Looks modern and improves engagement.

✓ C. Breaking News Banner

- Full-width red bar.
- Uses urgency color (red) to catch immediate attention.
- Exactly like real news portals.

✓ D. Hero Section (Large News Card)

- Occupies 70% width on desktop.
- Shows the biggest story of the day.
- Includes:
    - Big image
    - Category badge
    - Headline + excerpt
    - Date & time

✓ E. Secondary Cards (Right Side)

- Two equal small cards on right side.
- Good for highly-ranked but less important news.
- Mimics Hindustan's front page.

✓ F. Mobile Responsive Layout

- Large card becomes full width.
- All cards displayed vertically.
- Trending pills become scrollable.
- Navigation collapses into hamburger menu.

✓ G. Visual Hierarchy

- Bold red = high priority (breaking/category tag).
- Big image = top story.
- Small cards = secondary.
- Clean spacing improves readability.

---

4. Data Fetching Strategy (With Reasoning)

Chosen: `getStaticProps` (+ optional ISR)

Used in:

```
pages/index.tsx
pages/[category].tsx
pages/news/[id].tsx
```

✓ Why This Strategy?

1. Fast Performance (Pre-rendered HTML)
Pages load extremely quickly because they are generated at build time.

2. SEO Friendly
Search engines read static HTML easily.

3. Perfect for News Homepages
News must be fast but not necessarily real-time.

4. ISR (Incremental Static Regeneration) ***(optional)***
Allows auto-refresh every 5 minutes:

```
revalidate: 300
```

✓ Tradeoffs

Method	Pros	Cons
getStaticProps (Used)	Fast, cached, SEO, ISR	Not instant updates
getServerSideProps	Always fresh	Slower, more API hits
Client Fetch	Smooth UX	Poor SEO

Conclusion:
`getStaticProps + ISR` gives the best balance.

---

5. Component Explanation (Based on Your Folder Structure)

✓ `components/layout`

Component	Purpose
Header.tsx	Shows logo, main navigation, action icons
Navigation.tsx	Category navbar (Home, State, Country, etc.)
Footer.tsx	Footer content

✓ `components/common`

Component	Purpose
TrendingTopics.tsx	Horizontal pill-style trending topics

✓ `components/news`

Component	Purpose
FeaturedNews.tsx	Large Hero News Card on homepage
NewsCard.tsx	Reusable news card (horizontal & vertical)
NewsGrid.tsx	Grid for sections like “Top Hindi News”

✓ `pages`

Page	Purpose
`pages/index.tsx`	Homepage

Page	Purpose
`pages/[category].tsx`	Loads category-specific news
`pages/news/[id].tsx`	Dynamic article page
`pages/api`	Backend mock API routes (optional)
`_app.tsx`, `_document.tsx`	Global config, font loading

✓ **`data/mockData.ts`**
Stores all mock news items (120 articles × categories).

✓ **`types/index.ts`**
Contains TypeScript types for data models.

---

6. Data Model Explanation

NewsItem Type

```
export interface NewsItem {
  id: number;
  title: string;
  image: string;
  date: string;
  time: string;
  category: string;
  excerpt?: string;
  content?: string;
}
```

✓ Why this structure?

- Enough to display all necessary content on cards and detail page.
- Aligns with real news APIs (title, urlToImage, publishedAt…)
- Flexible for adding new fields later (author, location, tags…)

---

7. Challenges Faced (Realistic + Based On Your Errors)

❗ 1. Next.js Image Optimization Errors

NewsAPI images came from unlisted domains → Next.js threw errors.
Solution:
Added this in `next.config.js`:

```
images: {
  remotePatterns: [{ protocol: "https", hostname: "**" }],
}
```

❗ 2. JSON Serialization Errors

Next.js refused to serialize `undefined` fields returned from API.
Solution:
Convert all undefined → null in normalizer:

```
city: null,
state: null,
```

```
updatedAt: null,
```

❗ 3. Responsive Layout Issues
Hero card & sidebar alignment broke on mobile.
Solution:
Used Tailwind breakpoints:
```
lg:flex-row flex-col
md:grid-cols-3 grid-cols-1
```

❗ 4. Mapping NewsAPI Data
NewsAPI does not provide slugs or stable IDs.
Solution:
Use encoded URLs as IDs:
```
slug: encodeURIComponent(article.url)
```

---

8. Future Improvements

⭑ 1. Real Backend + Database
- Store articles
- Real slugs
- Avoid API limitations

⭑ 2. Search Feature
Global search across titles/excerpts.

⭑ 3. Infinite Scroll
Load more news dynamically.

⭑ 4. Dark Mode
Modern look + user preference saving.

⭑ 5. Better SEO
Dynamic meta tags, OpenGraph previews.

PART C – Testing and Edge Case Handling

This section describes how the application behaves under different data conditions and how potential errors are handled gracefully. Since the project uses **local mock data instead of an external API**, all tests were performed by modifying the mock data and testing how the UI responds.

---

1. Missing Image Handling

Scenario:

A news item does not contain an image or has an empty image string.

Expected Behavior:

- Display a **default placeholder image**.
- Prevent broken-image icons and layout shifts.

Implementation:

```
<Image
  src={item.image || "/placeholder.jpg"}
  alt={item.title}
  width={400}
  height={250}
  className="object-cover rounded-md"
/>
```

Result:

The UI remains clean, consistent, and visually stable.

---

2. No Articles in a Category

Scenario:

A category array in `mockData` is empty, e.g.:

```
categoryNews["Sports"] = [];
```

Expected Behavior:

- Show a fallback message: **“No news available in this category.”**
- Do not render empty cards or grids.

Implementation:

```
if (articles.length === 0) {
  return <p className="text-gray-500 text-center py-8">
    No news available in this category.
  </p>;
}
```

---

3. Extremely Long Headlines

Scenario:

Some news items contain titles that exceed two or three lines.

Expected Behavior:

- Text should wrap neatly without breaking the layout.
- Titles should be clipped if too long.

Implementation:

```
<h2 className="line-clamp-2 break-words">
  {item.title}
</h2>
```

Result:

Cards remain aligned, and the layout does not stretch or overflow.

---

4. Missing Optional Fields (Excerpt / Content)

Scenario:

Some mock news entries intentionally lack `excerpt` or `content`.

Expected Behavior:

- Show a fallback message when optional fields are missing.

Implementation:

```
<p>{item.excerpt || "No description available"}</p>
```

Result:

The UI always shows readable content even if some fields are missing.

---

5. Loading State Simulation

Scenario:

To test loading behavior, a delay was added to mimic API latency:

```
await new Promise(res => setTimeout(res, 1000));
```

Expected Behavior:

- Show a loading indicator until content appears.

- Prevent blank screens.

Implementation:

```
<p className="text-gray-500 py-6 text-center">Loading news...</p>
```

---

6. Error Handling for Invalid Category Routes

Scenario:

A user manually enters an invalid category URL, e.g.:
`/news/xyz-unknown-category`

Expected Behavior:

- Show a custom error or “Category not found” page.

Implementation:

```
if (!categoryNews[params.category]) {
  return <NotFound message="Category not found" />;
}
```

---

7. Responsive Layout Testing

The application was tested across screen sizes:

- **320px** (small mobile)
- **375px** (iPhone)
- **768px** (tablet)
- **1024px+** (desktop)

Checks Performed:

✓ Breaking News banner scales properly
✓ Trending pills scroll horizontally on mobile
✓ Hero card becomes full-width on small screens
✓ Grid turns into vertical list on mobile
✓ Typography remains readable

Result:

The layout adapts correctly for all device sizes.

---

8. General Stability with Local Mock Data

Since mock data is used:

- No API rate limits
- No network errors
- No unpredictable responses
- No need for error retries

This makes the application stable, consistent, and ideal for demonstration and evaluation.

✓ Final Summary

The application gracefully handles missing fields, empty data, long text, invalid routes, and loading delays.
All edge cases were tested using controlled modifications in `mockData.ts`. The UI remains stable and user-friendly under all conditions.

PART D – AI Use + Reflection

This project was completed primarily through my own coding, debugging, and design work. Only around **15% of the work involved assistance from AI tools**, mainly for support tasks, not core implementation.

Below is an honest and detailed reflection on how AI was used and how I ensured correctness.

---

1. Parts of the Assignment Where AI Was Used (≈15%)

a) Documentation Writing

AI helped in:

- Structuring Part B documentation
- Rewriting wireframe descriptions
- Organizing layout and design explanation
- Writing this Part D section

AI was used only to **format and refine text**, not to write code.

---

b) Folder Structure Planning

AI suggested:

- A clean folder layout
- Separation of components (`layout`, `common`, `news`)
- Placement of mock data and types

I used these suggestions but **implemented the full structure myself**.

---

c) Generating Mock Data

AI helped generate:

- A consistent mock dataset
- Multiple categories with sample news items

But I:

- Verified all values
- Fixed broken image URLs
- Adjusted categories and titles manually

---

d) Debugging Assistance

AI helped identify and explain:

- Next.js image domain configuration error
- Serialization issues with `undefined` fields
- Folder mismatch and mockData loading problems

However, the debugging and fixes were implemented manually.

---

2. Where AI Suggestions Were Wrong or Suboptimal

AI provided some suggestions that did **not work correctly**, such as:

a) Incorrect Next.js Image Config

AI initially suggested using:

```
domains: ["*"]
```

(which is invalid)

I researched and replaced it with:

```
remotePatterns: [{ protocol: "https", hostname: "**" }]
```

b) API Normalization Logic

AI's mapping logic produced:

- `undefined` values
- Missing fallbacks
- Incorrect slugs

I manually rewrote the mapping function and finally removed API usage entirely.

c) Dynamic Routing Issues

AI suggested slug-based paths using NewsAPI, which is impossible because NewsAPI gives no slugs.

I corrected this by:

- Using mock IDs
- Creating clean dynamic pages with predictable paths

---

3. How I Verified and Corrected AI Suggestions

✓ Manual Testing

I manually tested:

- UI layout
- Responsive design
- MockData consistency
- All pages with and without data

✓ Rewriting Code Myself

AI sometimes produced generic or non-optimized code.
I improved it by:

- Simplifying components
- Adding TypeScript types
- Cleaning unused imports
- Adjusting layout logic (especially mobile view)

✓ Cross-checking Documentation

AI-generated explanations were corrected by:

- Removing inaccuracies
- Adjusting to match my actual project
- Rewriting final content in my own words

---

4. Custom Modifications I Made Beyond AI Suggestions

a) Full UI Implementation

I built:

- FeaturedNews
- NewsCard
- NewsGrid
- Navigation
- TrendingTopics
  from scratch using TailwindCSS and Next.js.

b) Responsive Layout

Mobile and desktop breakpoints were completely hand-coded by me.

c) Clean TypeScript Models

I created:

- Types for mock news
- Category mapping
- Correct interfaces

d) Better Folder Architecture

I reorganized:

- Components
- Pages
- Data
- Types

in a much cleaner structure than what AI proposed.

e) Switching from API to MockData

I made the strategic decision to drop NewsAPI and maintain local mockData for stability.

Final Reflection

AI helped **only for supporting tasks**, not core development.
Around **85% of the work—including coding, debugging, layout building, UI design, responsiveness, logic, and routing—was done manually by me.**

AI mainly served as:

- A helper for documentation
- A debugging assistant
- A generator for sample mock data
- A reference for structuring ideas

All final code and implementation were independently written, tested, and refined by me.